Improving grid fault tolerance by means of global behavior modeling

Jesús Montes
CeSViMa
Universidad Politécnica de Madrid
Madrid, Spain
Email: jmontes@cesvima.upm.es

Alberto Sánchez
E.T.S. de Ingeniería Informática
Universidad Rey Juan Carlos
Madrid, Spain
Email: alberto.sanchez@urjc.es

María S. Pérez
Facultad de Informática
Universidad Politécnica de Madrid
Madrid, Spain
Email: mperez@fi.upm.es

***Abstract*—Grid systems have proved to be one of the most important new alternatives to face challenging problems but, to exploit its benefits, dependability and fault tolerance are key aspects. However, the vast complexity of these systems limits the efficiency of traditional fault tolerance techniques. It seems necessary to distinguish between *resource-level* fault tolerance (focused on every machine) and *service-level* fault tolerance (focused on global behavior). Techniques based on these concepts can handle system complexity and increase dependability.**

We present an autonomous, self-adaptive fault tolerance framework for grid systems, based on a new approach to model distributed environments. The grid is considered as a single entity, instead of a set of independent resources. This point of view focuses on *service-level* fault tolerance, allowing us to *see the big picture* and understand the system's global behavior. The resulting model's simplicity is the key to provide system-wide fault tolerance.

I. Introduction

Large scale distributed systems have paved the way to face complex, technical and scientific challenges that can not be solved with traditional systems, due to their huge computing and/or storage requirements. Initiatives such as BOINC [1], PlanetLab [2] or TeraGrid [3] and, more generally speaking, grid [4] or the recent cloud computing [5] provide computing and storage resources that can be scaled to a level difficult to imagine elsewhere. Nevertheless, in spite of their potential, building dependable grid systems is not an easy task.

For the purpose of providing fault tolerance to these systems, a deep knowledge about the behavior of each single element is usually required. However, the huge number of different resources makes it almost impossible to analyze and implement efficient policies on every one. Most of traditional and current grid management techniques are based on this approach [6]–[8], dealing with each independent resource's behavior separately. A good alternative could be simplifying the understanding of the whole system, studying it as a single entity instead of the set of elements that together form it. This abstraction would describe how the system globally works and simplify its management.

Our approach combines the use of self-adaptive techniques with a *single entity* vision of the grid in order to provide fault tolerance and increase dependability. Our approach is unique in that we use this single entity vision to focus on service-related global aspects.

The paper is organized as follows. Section II addresses fault tolerance issues, discussing the concept of failure on grid systems and our approach to model global behavior. Section III proposes a way to build dependable grid systems based on global behavior modeling. Section IV validates our approach. Section V shows different related work and section VI explains the final conclusions and outlines future research directions.

II. Fault tolerance issues in grid computing

Most grids are not only distributed in nature, but also heterogeneous, non-centralized and in most cases composed of non-dedicated resources. Providing fault tolerance in such complex systems is not a simple task. These properties, added to the fact that grids are large scale systems (and therefore they have a large number of resources), bring the problem to a new level, and it does not seem a matter of simply adapting existing distributed computing fault tolerance techniques.

The inherent complexity of grid systems makes the direct application of these techniques very difficult. In grid, heterogeneity, variability and decentralization are considered, in most cases, as system features. As a consequence, resources can unpredictably appear and disappear, network links can be temporarily or permanently interrupted, parts of the system can be overloaded without any control from the global system administrators and so on. These events are normally considered faults in traditional distributed systems, but in grid computing they are part of the system's typical behavior. Therefore, is not so clear if these events are faults or not. The lesser degree of cohesion of grids dilutes the concept of failure based on the loss or degradation of resources. Grids are commonly seen as an immense set of resources that provide a series of services. Their proper operation should be understood in terms of the quality of the services provided instead of the state of its resources.

A. Single entity vs. multiple entities

One of the most puzzling aspects of grid systems is that they are considered as single elements in theory but, when it

comes to practice (specially in management related issues), they are treated as a set of independent, loosely related, elements. It might be argued that these systems are no simple ones and their great complexity makes necessary to look after every one of its parts. However, it could simply be a matter of perspective.

To illustrate this idea, it is interesting first to analyze the case of a single desktop computer. This apparently much simpler system is commonly regarded and managed as a single device but, in fact, it is composed of a large set of sophisticated elements that cooperate. Elements like CPUs, memory and its controllers, video cards, hard drives, network interfaces and so on have distinctive functionalities and are technologically complex, but are seen as parts of a single entity, instead of a set of heterogeneous resources. This change of perspective is due to the use of high-level tools (basically the operating system) that provide an abstraction layer between the real, heterogeneous and complex hardware and the user. Several generic parameters are defined, such as CPU load or network usage, in order to express the system state in a standard manner. Even though this abstraction carries some loss of information, it allows the managing techniques to be standardized, regarding all desktop computers by the same parameters. Considering fault tolerance, generic procedures are developed in the same way.

If this concept is applied to grids, it becomes clear that the proper tools for making this abstraction are yet to be established. Grids are still considered as a set of parts, instead of the sum of them. In consequence, the management tools inherit the complexity of the system, not allowing synergy to take place.

B. Fault tolerance in grid systems: resource-level vs. service-level

Distinguished by its point of view, fault tolerance techniques in grid systems can be split into two categories: *resource-level* and *service-level*. In order to optimize performance and increase system dependability the correct combination of these two types of techniques should be applied. However, some important aspects must be considered.

Resource-level fault tolerance involves the application of standard fault tolerance techniques in each and every one of the resources in the system. This might seem pretty straightforward, but carefull consideration reveals that most of typical grid characteristics could limit its efficiency, as is now explained. The heterogeneous and non-dedicated nature of the system increase complexity, but it is the non-centralized aspect the one that becomes the great difficulty. In many cases, the global management system has so limited control of each resource that the only suitable solution seems to increase redundancy.

Service-level fault tolerance, on the other hand, deals with system-wide policies aiming to increase dependability of the services provided. This is particularly important in utility computing systems, where the quality-of-service (QoS) is the key factor. However, as the fault tolerance policies have to deal with the whole system, it is important to find ways to efficiently manage this complexity. It is also important to understand that, as the nature of the system is different from *resource-level* fault tolerance, the terms in which this fault tolerance is expressed certainly differ.

In *resource-level* fault tolerance basic concepts such as fault or failure are directly inherited from traditional distributed systems. Events such as a machine turning unexpectedly off or the temporary loss of a network link are clearly regarded as faults. But in a non-dedicated, non-centralized distributed system like a grid, each partner that shares resources keeps full control over its property (computing nodes, storage elements, network links, etc). Resource providers can change the state of its own resources, without consent from the grid global management. For instance, some machines could be turned off, originating an event that would be probably considered a fault in traditional distributed systems fault tolerance. But in grid systems these events are by no means considered as undesirable or unexpected. They are more likely accepted situations that not only may, but will occur as part of the natural evolution of the grid. Therefore *service-level* fault tolerance can never regard them as faults.

Service-level fault tolerance should focus on QoS issues and global behavior. It can benefit from a representation of the grid global state in a service oriented form. This would become a behavior model based on globally service-relevant states instead of the multiple specifics of each resource. This representation not only seems ideal for *service-level* fault tolerance, but also provides the abstraction layer mentioned in the previous subsection. With such a model, grid management tools could finally have the previously mentioned single entity perspective, incorporating the system's complexity without being overwhelmed by it. This could also take *service-level* fault tolerance a step further, better understanding and improving the systems behavior and dependability.

C. GloBeM: Global behavior modelling of grid systems

A global behavior model would provide the abstraction layer that finally makes the single entity point of view possible in grid computing. In order to do so, this model must have certain characteristics:

- **Specific state definition:** State characteristics and transition conditions should be unambiguously specified.
- **Stability:** The resulting model must be considerably consistent with the system behavior over time.
- **Simplicity:** The resulting model should be understandable and provide basic and meaningful information about the systems behavior.
- **Relevance to service:** The model states should be related to the system services.

In [9], [10] we introduced a methodology for creating this kind of model. It is strongly based on traditional data mining tools, and also in more advanced knowledge discovery techniques such as virtual representation of information systems [11]. It uses monitoring information gathered from a grid to produce a finite state machine that represents its behavior (please refer to the above mentioned paper for further details about this technique). This methodology is now called GloBeM[1]. The resulting model produced by GloBeM becomes an abstraction layer on top of the grid infrastructure. However in [9] the global behavior model is presented in a general way, and no specific application is indicated. Our approach takes advantage of this single entity point of view in order to improve fault tolerance in grid systems. Our research focus on a new approach to *service-level* fault tolerance, based on a single entity vision of the grid.

III. Service-level fault tolerance based on global behavior modelling: FIRE

Combining the ideas of single entity view and *service-level* fault tolerance, an autonomic management framework called *FIRE*[2] has been developed. The basic idea behind FIRE is that, if several states can be distinguished within a grid (the way GloBeM does it), different system behavior should be obviously expected for each one of them. It seems reasonable to assume that not all fault tolerance techniques would be optimal for every state. Therefore, if a set of compatible management policies are available, it would be essential to identify which one is most adequate for each state and provide the necessary mechanisms to switch between them when the system shifts from one state to another.

If we consider data replica allocation, for instance, several different allocation policies could be used, depending on the system state. Some polices could be optimal when the network connections are heavy loaded, others when the CPU usage is very unbalanced, etc. Dynamically selecting the adequate replica allocation policy for each state could strongly improve the overall system's performance (assuming that all policies are compatible among themselves). FIRE's main purpose is exactly that: to serve as a simple but effective automated policy selector, based on a GloBeM's finite state machine model of the system's behavior.

FIRE monitors the system and performs a representation of information in a similar way the global behavior modelling procedure does. Then it determines the current system state, using a previously provided finite state machine model.

[1]**Glo**bal **Be**havior **M**odeling

[2]The acronym *FIRE* stands for "**F**IRE **I**sn't just a **R**eplication **E**nvironment". The reason behind that name is that the system was originally conceived as a data replication policies manager. Nowadays it has grown beyond that, to deal with different types of *service-level* autonomic management.

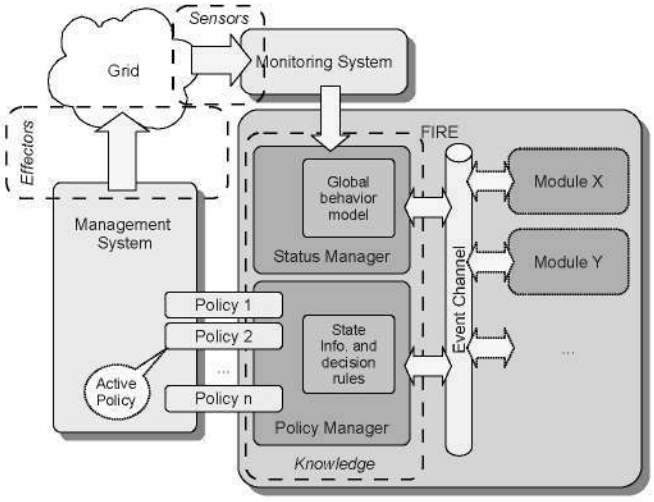

Figure 1. FIRE's architecture

It then activates the correct policy for the current state. The corresponding policy for each state must also be provided to FIRE as part of its configuration.

The policies controlled by FIRE can be of any kind. Typical examples of this are a set of interchangeable job scheduling policies or the above mentioned data replication policies. FIRE has to communicate with the proper management subsystem (job scheduler, data manager, etc) in order to activate the proper policy.

A. Architecture of FIRE

Figure 1 illustrates the FIRE's architecture. It has been designed to provide an extensible, adaptive, autonomic framework for grid management. From an autonomic point of view, the system must present the following elements:

- **Sensors** (*the eyes*): These are the elements that gather information about the grid evolution and behavior. To this purpose FIRE takes advantage of a grid monitoring service.
- **Effectors** (*the hands*): These are the elements that perform the actual grid management, following a specific policy or set of policies. The specific characteristics of these effectors change depending on the grid services and applications. In a data grid, for instance the effectors would be those software tools in charge of the management operations such as data allocation, load balancing, etc.
- **Knowledge** (*the brain*): This the autonomic system's core. It contains the necessary information and capabilities to perform four basic tasks: *a*) *Monitor* (by means of the sensors): This makes the system aware of its own state. *b*) *Analyze*: This makes possible to understand the system's state in terms of the behavior model in use. In the case of FIRE, this analysis is based on a global

behavior model generated using the GloBeM methodology. *c*) *Plan*: Once the system's behavior has been observed and understood, the appropriate management decisions are made, in order to self-adapt to the current conditions. *d*) *Execute* (by means of the effectors): The planned decisions are executed. These four tasks are the basis to provide autonomic capabilities to grid management mechanisms. FIRE is focused on these aspects, providing the *knowledge* element in the grid autonomic management.

At it can be seen in Figure 1, FIRE itself does not stand as a complete autonomic solution, but as a basic framework to incorporate autonomic capabilities to a grid management system. From an architectural point of view it is designed around an standard **event channel**[3] in order to naturally increase its modularity and simplify its adaptation to different management problems. FIRE has three main elements: the *event channel* and the two main modules, connected through it (the *Status Manager* and the *Policy Manager*). It may contain also some other additional modules, in order to add new functionalities.

The *Status Manager* gathers monitoring information from the system resources. Then, with the use of the GloBeM finite state machine model, determines the current state and notifies it tho the *Policy Manager*. The *Policy Manager* receives the current state and determines which policy is to be activated. It then activates the policy and notifies this fact to the corresponding management subsystem.

FIRE requires of some initial configuration (providing the finite state model and the corresponding policies) and therefore it is not a completely autonomous system. However, it makes the administration work much easier. Once the finite state machine is automatically generated, the system administrator only has to decide which policy fits better in each state, in order to increase dependability. Using FIRE as a basic management framework, system administrators can effectively manage a large scale distributed system without being overwhelmed by its complexity. The service oriented behavior model is the key to "see the big picture" and focus on global dependability and QoS.

IV. System evaluation

In order to evaluate the benefits of the FIRE framework, a set of experiments has been performed using the grid computing simulator GridSim [12]. GridSim is a widely accepted, powerful simulation tool for this kind of systems. It offers the possibility of realistically simulate hundreds of machines and clients interconnected through a complex network. In a work like this one, where new management techniques are being tested, it is almost impossible to experiment on real grid systems (such as the EGEE project [13]) due to the high level administrative capabilities required. Therefore, a simulation seems to be proper first step, specially if it is done with the adequate tools.

A. Experiment objectives

It has been said that FIRE can address many different problems, depending on the management system or systems it is working with and the set of policies provided. One of the most common uses of large scale distributed systems in general and grid computing in particular is the execution of CPU-intensive distributed applications (large distributed computations, high-performance computing, etc.). The system's distributed nature allows to run multiple jobs in different resources, but to efficiently do it, an adequate job scheduler is required. The capabilities of this scheduler can almost entirely establish the system's dependability, and therefore they are of maximum importance.

For these tests, a grid job execution service was simulated. Since FIRE addresses *service-level* fault tolerance, the basic parameters of the experiment must be service oriented. In this sense, job failure rate can be used to measure the QoS provided by this service from a fault tolerance point of view. Thus, randomly generated jobs were submitted to the simulated grid through a job scheduler and their failure rate was measured. To determine a job failure, a time deadline was established for each of them, based on its time of submission and job size. A job failure, in consequence, could be originated by a resource crash, a network overload, etc. The objective of these experiment was to show how FIRE, with the appropriate policies, can increase system's dependability by lowering the job failure rate.

B. Simulated scenarios

In order to produce a as much realistic as possible simulation of a real grid, performance statistics and job accounting information from the EGEE project was used [14]–[16]. The EGEE connects more than 70 institutions in 27 European countries to construct a multi-science grid infrastructure for the European Research Area. Three scenarios were designed, each of them with certain defined characteristics:

- **Randomized resources:** Computing resources were slightly randomize to obtain certain heterogeneity. The number of resources was fixed for each scenario, but the computing power of each of them was randomly generated. Each resource may have one or two machines, each of them with one or two processing elements (CPUs). The power of each processing element was randomized between 1000 and 5000 MIPS.
- **Randomized clients:** Each scenario had a different number of clients. These clients function was to randomly generate different types of load, generating

[3]This is a standard software design technique where communication between modules is carried out by a central event manager. A set of events are specified and the different modules can act as event publishers and/or event subscribers. This structure strongly simplifies the introduction of new modules on the system.

Scenario name	20R	50R	100R
Num. resources	20	50	100
Num. CPU load clients	10	25	50
Num. Net load clients	10	20	20

Table I
TEST SCENARIOS

continous (but not constant) network traffic and CPU load. Different levels of random CPU load and random network traffic were injected in order to simulate the uncontrollable changes in the system.

- **Resource failures:** Each resource had a random chance of failure. These were isolated failures that temporarily disconnected the resource from the system randomly affecting its composition. The failure parameters (probability of failure and duration of failure) were adjusted to fit real job failure rates observed on the EGEE (this is explained in more detail below).
- **Job dispatcher:** In each scenario there was a job dispatcher that represented the grid service, with a queue of randomly generated jobs. Each job had three randomly generated parameters: the job **computing size** (between 100 and 100000 millions of instructions) **data input size** (between 0 and 50 MB) and **data output size** (also between 0 and 50 MB). These are the three basic job parameters used by GridSim.

Table I shows the different parameters established for each one of the tree simulation scenarios (called 20R, 50R and 100R). All tests simulated 30 days of execution of these systems.

Since FIRE aims at improving fault tolerance, one of the main aspects that must be considered in order to perform a realistic simulation of this grid is the job failure rate. As EGEE was used as a reference for the simulated scenarios, it was important o reproduce the same failure rates observed in the real system. The above mentioned references show that this parameter oscillates due to many factors, but it is usually around 16%. For the simulated scenarios, it was decided to generate a basis job failure rate of 16%, and then show how the use of FIRE lowers this rate. This value includes any kind of job failure, both generated by resource failures and/or network problems.

C. Behavior model

Prior to performing the FIRE tests, an initial configuration was designed for comparative purposes. This configuration used FCFS (First-come, first-served) as the only job scheduler policy, so jobs were always strictly dispatched in the order they arrived. It was executed on all three test scenarios and a behavior model was generated using the methodology previously mentioned. The resulting finite state machine can be seen in Figure 2. It is important to remind that this is an automatically generated model, and the state analysis took

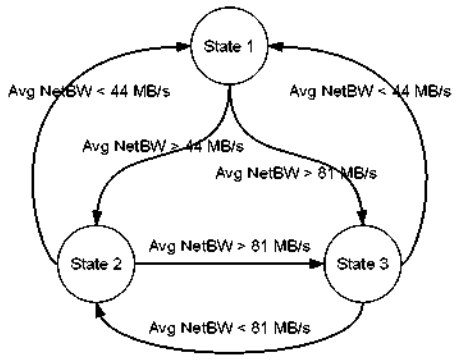

Figure 2. Global behavior model of the test scenario

place after its construction. This ensures that the resulting states are based only on the behavior information monitored and not on any system administrator's personal assumptions.

The three observed states are:

- **State 1:** It is characterized by a low average network bandwidth (below 44 MB/s), mostly due to network overload.
- **State 2:** It is characterized by a medium average network bandwidth (between 44 and 81 MB/s). It seems to represent the medium load state of the grid.
- **State 3:** It is characterized by a high average network bandwidth (over 81 MB/s). This represents a barely loaded grid, where the network can be used at full capacity.

From a service point of view, state 1 seems to be the most problematic, as low network bandwidth can make the data input and output transfer times longer and therefore increase the job's failure probability. State 3, on the other hand, seems like the best one, as the high network bandwidth guarantees fast data transfers. State 2 certainly is in an intermediate point.

In order to increase dependability, FIRE needs a set of policies adapted to each state. In this case, a set of job scheduling policies was configured, aimed to improve system's dependability. To make the example easier to understand, the chosen policies were very simple but still effective:

- **Policy A** gave a higher priority to jobs that had small input and output data. This reduces the chances of job failure when the network is slow. This policy was configured for **state 1**.
- **Policy B** gave a higher priority to jobs that had large input and output data. This was specifically designed to take advantage of times when network bandwidth is very high. The idea is to execute the heavier jobs when their success chance was higher. Obviously this policy was configured for **state 3**.
- **Policy C** dispatched jobs in strict arrival order (FCFS).

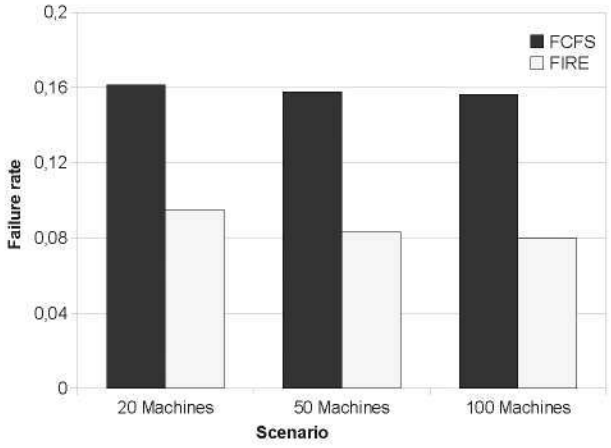

Figure 3. Job failure rate for each scenario and policy configuration

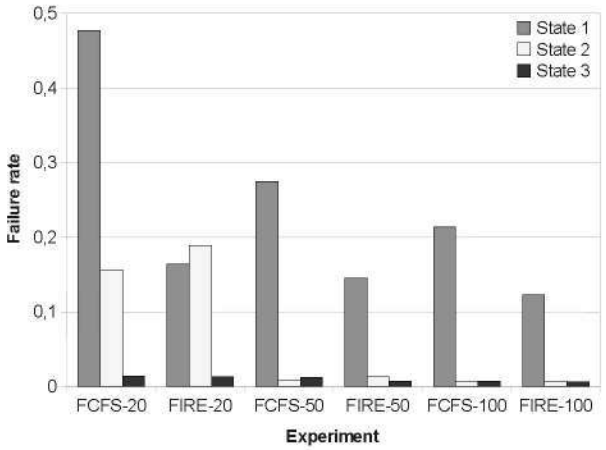

Figure 4. Job failure rate for each state in each experiment

It was configured for **state 2**.

D. Simulation results

Each scenario (20R, 50R and 100R) was simulated using the basic FCFS scheduling policy and the special multi-policy scheduler controlled by FIRE. 16 simulations (each of them using different random seeds) were performed for each scenario and scheduler system, giving a total number of 480 days of simulated time for each experiment (every execution simulated 30 days).

The average job failure rate results for each experiment can be seen in Figure 3, grouped by scenario. As it has been said, the job failure rate for the FCFS configuration was fixed to 0.16, in order to produce a value observed on a real grid (EGEE, in this case). The FIRE configuration, as it can be seen, clearly reduces the job failure rate in every experiment.

A more detailed analysis is displayed in Figure 4. In this case, every scenario configuration is displayed separately but, in each of them, separated failures rates are displayed for each state. It is clear now that the state where most of job failures occur is state 1. The multi-policy based FIRE configuration succeeded in lowering this state failure rate.

As a curious detail, it can also be seen in Figure 4 that the use of the multi-policy FIRE configuration very slightly increases the state 2 failure rate. Although this does not affect the overall result, it is interesting to provide an explanation for this phenomenom. It is important to remember that the associated policy to state 1 (policy A) increases the small jobs priority, making them more likely to be executed. In consequence, this makes that most small jobs are executed while this policy is active, and when the system returns to state 2, the average job size in the queue is certainly higher. Therefore the jobs executed during state 2 are generally bigger than in the FCFS configuration and their chance of failure is higher. This slightly increases state 2 failure rate. Even though, this increase has little effect in the global failure rate.

V. Related Work

Our proposal aims at providing adaptive fault tolerance, based on the grid behavior. Several approaches have arisen with the idea of modelling and characterizing the behavior of a grid or large distributed system.

One first step in the characterization of any computer system is the use of benchmarking [17], which allows for the analysis of the performance behavior of a system when different workloads representing the whole spectrum of possible loads are applied. These workloads are the result of running a benchmark program with specific inputs and configuration parameters. Once the benchmark program has been run, a performance model of the system is obtained. Unlike grid benchmarking, our approach models the behavior of a grid according to data obtained from the monitoring of the system in real scenarios and with real users, applications and resources.

Other different works have appeared for modelling a grid. Bratosin et al. in [18] provide a formal description of grids by using Colored Petri Nets (CPN). This model is used for simulation of grids. Unlike this model, our proposal does not simulate a grid but it models the grid with the aim of simplifying the knowledge about its behavior and helping to make decisions at run-time.

In the same line than the previous work, Nemeth et al. [19] present a formal definition of a grid by means of the use of Abstract State Machines (ASM) [20], a mathematical framework for analysis and design of systems. The model obtained in this work enables distinguishing semantically conventional distributed systems from grids. However, this

definition is based on an idealistic grid and only considering its qualitative characteristics.

On the other hand, different research works have addressed the problem of fault tolerance in grids and large scale systems. The dynamism and lack of centralized control of these systems make the efficient application of fault tolerance techniques very difficult. Moreover, a redefinition of the fault model in these environments is needed. An enhanced fault model for grids has been developed within the e-Demand project at the University of Durham [21]. This work improves the fault model of a traditional distributed system, including new type of faults, namely potential life-cycle, metered access, interaction, timing, and omission faults. An approach for grid applications combining replication-based fault tolerance and dynamic prioritization and scheduling is also provided in this project. Unlike this work, our proposal changes the perspective from a resource-based model to a system-based model, redefining completely the fault model. This redefinition simplifies the application of fault tolerance techniques in grids.

Other authors present fault tolerant mechanism for grid applications. A mechanism for divide-and-conquer grid applications is shown in [22]. This work focuses exclusively on these applications, limiting its scope. Luckow et al. [23] describe a more generic fault tolerance approach for grid applications, although their solution is more oriented to parallel MPI applications, whose behavior in a grid is clearly limited by large latencies. Our approach provides a higher level model for providing fault tolerance in these systems.

VI. Conclusions and Future Work

Grid systems are suitable in high demanding scenarios in which other computing solutions have traditionally failed. However, one of the weakest aspects of these systems is its dependability. The system's complexity, higher probability of failure and dynamism make it difficult to achieve a dependable state, considering the following definition of dependability: *"the dependability of a system is the ability to avoid service failures that are more frequent and more severe than is acceptable"* [24].

Our approach provides fault tolerance based on a global behavior model of the grid that simplifies the complex vision of this kind of systems. This new approach makes it easier to apply suitable management techniques, which improves largely the system's dependability. On the one hand, the finite state machine used by our proposal simplifies the making decision tasks over the system. On the other hand, as Section IV shows, the FIRE framework built within our work enables the proper application of management policies. Consequently, this improves significantly the system's dependability.

As future work, we are planning to validate our proposal in a real system (insted of a simulated grid). Although the simulator GridSim models realistic scenarios, it would be desirable to further validate the properties of our model and our framework with data from a real scenario.

Furthermore, we plan to extend this scenario to other management systems, such as the data manager. Thus, we intend to apply our model in order to enhance the system's dependability regarding to data access services.

Finally, if we could predict the future behavior of the grid, we would be able to make better decisions oriented to improve not just current but also future conditions. We are planning to apply time series analysis techniques in order to predict the system's future state and this way improve grid autonomic management and dependability.

References

[1] "BOINC, accessed Feb 2010." [Online]. Available: http://boinc.berkeley.edu/

[2] "PlanetLab: An open platform for developing, deploying, and accessing planetary scale-services, accessed Feb 2010." [Online]. Available: http://www.planet-lab.org/

[3] "TeraGrid, accessed Feb 2010." [Online]. Available: http://www.teragrid.org/

[4] I. Foster, "What is the Grid? A Three Point Checklist," *Grid Today*, vol. 1, no. 6, Jul 2002. [Online]. Available: http://www.gridtoday.com/02/0722/100136.html

[5] "Twenty-One Experts Define Cloud Computing, accessed Feb 2010." [Online]. Available: http://cloudcomputing.sys-con.com/node/612375/print

[6] R. Buyya, D. Abramson, and J. Giddy, "Nimrod/g: an architecture for a resource management and scheduling system in a global computational grid," vol. 1, 2000, pp. 283–289 vol.1. [Online]. Available: http://dx.doi.org/10.1109/HPC.2000.846563

[7] K. Krauter, R. Buyya, and M. Maheswaran, "A taxonomy and survey of grid resource management systems for distributed computing," *Software Practice and Experience*, vol. 32, no. 2, pp. 135–164, 2002.

[8] M. Siddiqui and T. Fahringer, "GridARM: Askalon's Grid Resource Management System," in *Advances in Grid Computing - EGC 2005 - Revised Selected Papers*, ser. Lecture Notes in Computer Science, vol. 3470. Amsterdam, Netherlands: Springer Verlag GmbH, ISBN 3-540-26918-5, June 2005, pp. 122–131.

[9] J. Montes, A. Sánchez, J. J. Valdés, M. S. Pérez, and P. Herrero, "The grid as a single entity: Towards a behavior model of the whole grid," in *OTM Conferences (1)*, ser. Lecture Notes in Computer Science, R. Meersman and Z. Tari, Eds., vol. 5331. Springer, 2008, pp. 886–897.

[10] ——, "Finding order in chaos: a behavior model of the whole grid," *Concurrency and Computation: Practice and Experience*, p. In press., 2010.

[11] J. J. Valdés, "Virtual reality representation of information systems and decision rules," *Lecture Notes in Artificial Intelligence*, vol. 2639, pp. 615–618, 2003.

[12] R. Buyya and M. M. Murshed, "Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing," *CoRR*, vol. cs.DC/0203019, 2002.

[13] "Enabling Grids for E-sciencE (EGEE) project, accessed Feb 2010." [Online]. Available: http://www.eu-egee.org/

[14] "EGEE - Alice Job Summary, accessed Feb 2010." [Online]. Available: http://dashb-alice.cern.ch/dashboard/request.py/jobsummary

[15] "EGEE - Atlas Dashboard, accessed Feb 2010." [Online]. Available: http://dashb-atlas-prodsys-test.cern.ch/dashboard/request.py/summary

[16] "EGEE - Atlas Job Summary, accessed Feb 2010." [Online]. Available: http://dashb-atlas-job.cern.ch/dashboard/request.py/jobsummary

[17] J. J. Dongarra and W. Gentzsch, Eds., *Computer benchmarks*. Amsterdam, The Netherlands, The Netherlands: Elsevier Science Publishers B. V., 1993.

[18] C. Bratosin, W. M. P. van der Aalst, N. Sidorova, and N. Trcka, "A reference model for grid architectures and its analysis," in *OTM Conferences (1)*, ser. Lecture Notes in Computer Science, R. Meersman and Z. Tari, Eds., vol. 5331. Springer, 2008, pp. 898–913.

[19] Z. N. Németh and V. Sunderam, "A formal framework for defining grid systems," in *CCGRID '02: Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*. Washington, DC, USA: IEEE Computer Society, 2002, p. 202.

[20] Y. Gurevich, "Evolving Algebras: An Attempt to Discover Semantics," in *EATCS Bulletin*. European Assoc. for Theor. Computer Science, Feb. 1991, vol. 43, pp. 264–284.

[21] P. Townend and J. Xu, "Fault tolerance within a grid environment," in *In Proceedings of the UK e-Science All Hands Meeting 2003*, 2003, pp. 272–275.

[22] G. Wrzesinska, R. V. van Nieuwpoort, J. Maassen, T. Kielmann, and H. E. Bal, "Fault-tolerant scheduling of fine-grained tasks in grid environments," *International Journal of High Performance Computing Applications*, vol. 20, pp. 103–114, 2006.

[23] A. Luckow and B. Schnor, "Migol: A fault-tolerant service framework for mpi applications in the grid," *Future Generation Comp. Syst.*, vol. 24, no. 2, pp. 142–152, 2008.

[24] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *Dependable and Secure Computing, IEEE Transactions on*, vol. 1, no. 1, pp. 11–33, Jan.-March 2004.